DOCUMENT RESUME

ED 129 271 IR 004 053

AUTHOR Biggs, Donald E.
TITLE The Computer Subroutine in Information Handling.
PUB DATE 11 Oct 76
NOTE 22p.

EDRS PRICE MF-$0.83 HC-$1.67 Plus Postage.
DESCRIPTORS Computer Programs; *Information Processing; Libraries; *Programing
IDENTIFIERS *Subroutines

ABSTRACT

Generalized computational subroutines can reduce programing repetitions and wasteful computer storage use. The most useful are those that are flexible enough to handle a wide variety of situations. Subroutines may have details open to change in order to blend into the main program. They may be built into the computer library or supplied by the programer as part of this program deck. Library subroutines have limiting specifications and ranges, entry and exit commands and addresses which should be designed to fit into any program. Operating routines, the most prominant type of which are the input-output sets or "packages," differ from other subroutines in that they partially control the computer and return control to the main program when their work is completed. Basic subroutines of particular value in a chemical library include basic arithmetic computations, function evaluations, numerical analyses plus collating and sorting programs. Service routines--compilers, assemblers, debugging and machine testing programs--are also useful. (KB)

THE COMPUTER SUBROUTINE IN INFORMATION HANDLING

Donald E. Riggs
Director of Auraria Libraries
(Community College of Denver-Auraria, Metropolitan State College and University of Colorado at Denver)

One of the most used and most abused parts of computer technology is the subroutine. Many times one finds it necessary to repeat the same set of instructions at different points in a computer program. For example, if one is dealing with complex quantities, he will find that most computers do not have direct commands which form sums, products, etc. Each time he wishes to form a product, it will be necessary to carry out a certain fixed sequence of additions, multiplications, and groupings which will produce the desired result. All such products will be formed in an identical manner, and it will be only the operands which differ. A possibility for handling this problem is to incorporate the needed instructions into the main program, simply repeating wherever desired.

If a set of n commands is required to form and store a product of two complex numbers, one might arrive at a program storage pattern possibly named complex product code. This is probably the technique which is most conservative of machine time, but is wasteful of storage capacity, since it requires one to duplicate what is essentially the same code in various storage locations. It is also wasteful of effort, since the codes are a function of storage location and one cannot exactly duplicate

them in each part of the program. It will be essential to modify each set of n instructions to make it suitable for its final location.

A better scheme is to write a single code which will form the complex product of any two designated complex numbers, and store it separately from the main program. Subsequently, each time one is required to form such a product, he will select this code, and, on its completion, return to the main program. Such a code is called a subroutine, or subprogram. It is an auxiliary routine used in conjunction with, but not as part of, the main routine or main program.

THE PURPOSE

It was the purpose of this study (1) to reveal the importance of computers' subroutines relative to the handling of information; (2) to relate the use of subroutines to a computerized chemical library; and (3) to provide an overall view of the effect of subroutines upon the daily activities of libraries and information centers possessing computers.

DEFINITION

Subroutine. Theoretically, anything that is written which makes a "positive" contribution to the programming field could be considered a subroutine. From a working standpoint, however, in this study subroutine means more. To be adequately defined, it must meet the following requirements:

1. It must make a positive contribution to the existing program library, for the basic programmer's vocabulary is thus

increased.

2. It must be general enough so that it will be used by many programs and often used many times within the same program.

3. It must represent a wise choice of the total number of functions it is capable of carrying out, because one cannot permit its flexibility to make it difficult or inefficient to use in any specific application.

ADVANTAGES AND DISADVANTAGES OF THE SUBROUTINE

A. ADVANTAGES

One of the major advantages of subroutine usage is the saving in time. This saving occurs in the calendar time of preparing a program for a computer and in the actual computing time.

A thoroughly tested subroutine can be regarded as correct. Therefore, if subroutines are employed in a newly developed program, their presence helps to home in on errors developing within the main program.

By increasing the vocabulary through the use of subroutines, the programmer is provided with a stimulus to new ideas and techniques. In any language there is an important correlation between the comprehensiveness of the vocabulary and the formation of ideas. With this greater variety comes greater flexibility.

The many fields employing computers have specialists and experts who are applying their knowledge to the creation of outstanding subroutines (Warheit, p. 486). A large library of subroutines from all fields has become available and is still rapidly growing. Such a wealth of output gives a tremendous impetus to

computer design, for the subroutines of today may well become the foundation for the basic vocabulary of the advanced computers of tomorrow.

What can this library of subroutines do for a new computer user? It is possible for a new computer organization to start production quickly, with a minimum of waste and with a minimum staff of professional programmers and machine operators. It is sometimes possible to go into some production without having to create wholly new programs, basing a system entirely upon available subroutines. Just as the basic instructions are the building blocks of the subroutines, so are the subroutines the building blocks of such programmed systems. In a similar manner, one has the ability to chain his subroutines (i.e., one can have subroutine within subroutine within another subroutine). Therefore, one can increase his programming system in depth as well as in breadth by employing orderly procedures.

B. POSSIBLE DISADVANTAGES

Possessing all the aforementioned advantages, what are the practical difficulties which could substantially reduce the effectiveness of the subroutine? In broad terms, the pitfalls are misuse, overuse, misdirection, carelessness, poor logic, the unanticipated error, uncalled-for duplication (including creation of worn-out discarded methods), and misleading or unavailable descriptions of working subroutines.

Too often, in an attempt to produce "new" methods or materials an installation or programming group will tend to value

quantity over quality. No matter how impressive their aims may have been, they make no real contribution. Today, however, the production of quality is becoming much more critical where we now have a workable foundation of techniques and applications. It is simply becoming more difficult to formulate good ideas and techniques which are absolutely new.

A much more wasteful, though less frequent, occurrence is the production of a major system, such as an assembly system, which incorporates ideas and techniques which have already been proved inadequate. This waste of time, talent and capital is something that should be eventually rooted out completed.

Often an excellent subroutine is not used because only a poor write-up, or perhaps no write-up at all, exists. Many fine pieces of work are buried by their write-ups; it is amazing how many misconceptions and misdirections can occur in such a description.

Next to improper write-ups, the most irritating and expensive difficulties are caused by errors in the subroutines themselves. One type of error, the uncalled-for error, will always exist to some degree. Poor use of flow diagrams, poor training, poor supervision, or just poor attitude will cause these errors.

The particular error that probably causes the most trouble is the unanticipated error. Especially in complex subroutine systems where one has a great variety of data that must be handled, a particular data combination may occur that is not treated correctly. Such a data combination may not be encountered during months of use. What happens when such an error occurs depends

upon how careful the programmer was in setting up error routines within the system.

Other times there is a tendency to use a subroutine where simple basic programming is much more effective. Improper use of subroutines do not weigh in their favor.

THE SUBROUTINE LIBRARY

With the production of subroutines increasing, it becomes much harder for the individual programmer to keep up with all the developments. It requires considerable effort to be familiar with so many routines to the point of being able to use them all properly. Actually, it is generally unnecessary for a programmer to be this proficient. How, then, does he decide which subroutines sound or look interesting enough for him to pursue them further? This responsibility is normally delegated to the librarian. The librarian is usually someone with a fair knowledge of the computer (or computers) being used in the organization, and able to understand technical terminology to the point of properly classifying each subroutine (Computers, p. 58).

Is it an input-output type, an arithmetic type, or perhaps a technique for solving a special set of equations? The computer organizations help in this area by submitting abstracts of the subroutines to all installations. The librarian normally prepares lists of new subroutines together with brief descriptions for distribution within the organization. He answers questions concerning the availability of subroutines meeting certain qualifications and is responsible for verifying all new subroutines

for accuracy. If he does not understand particular techniques or applications, he may assign them to knowledgeable programmers. He is also responsible for keeping current copies of all subroutines for immediate usage.

When a computer has magnetic tape as one of its inputs, certain subroutines are kept on this medium. Usually a selection of the most-used subroutines is placed on a special-purpose magnetic tape, which is kept up to date by the librarian. During the assembly-program procedure, when the programmer's language is being translated into its equivalent machine language, the assembly program recognizes a special pseudo-instruction informing it to retrieve the desired subroutine from the magnetic tape. Automatically the subroutine will combine with the main program that is being assembled.

One does not put all subroutines on the library tape because too much time would, therefore, be consumed in searching the tape. If a less popular subroutine is being used which is not found on the tape, then one can obtain a copy of the routine on punched cards from a file maintained for this purpose. Cards, however, do not provide the same degree of convenience or speed as the tape.

BASIC CONCEPTS AND PRINCIPLES

A. GENERALIZING A SUBROUTINE

The most useful subroutines in the library are the ones that will handle the widest variety of situations. A routine for

solving a system of precisely five simultaneous linear equations is far less useful than one that will handle any number of equations from two to thirty. Such flexibility is somewhat offset by the fact that a general routine of this kind may not be as efficient as one written to accommodate five equations, no more and no less. If an installation is expected to run four equations one day, twenty the next, etc., flexibility is to be preferred. On the other hand, if a large fraction of the time is spent solving systems of a fixed size, it is worthwhile to develop a specific program to deal with that number of equations (Wrubel, p. 91).

Many times with a small amount of effort a program can be generalized to include other useful cases. For example, the formula for sinh x is

$$\sinh x = \frac{e^{x} - e^{-x}}{2}$$

which differs from cosh x only in having a minus sign in the numerator. One can generalize his previous subroutine to compute either one.

B. OPEN SUBROUTINE

As already mentioned, the subroutine is comprised of a number of instructions used as a unit to perform a given function. One method of using subroutines is to insert this unit at every point in the program where it is needed to have this particular function performed. As one makes the insertion, he must modify certain instructions in the routine in order to adapt it to its

particular use at that point. One might, for example, have to modify some addresses in order for them to refer to the proper data. In other words, the details of the subroutine must be open to scrutiny in order to make it a part of the main program. Because it is thus open to change, this type of program is often called an open subroutine. Subroutines do not have to be in closed form (Scott, p. 400).

The open subroutine loses it separate identity once it has been inserted into the program. It simply becomes another set of instructions. Also, it is necessary to recopy such a subroutine each time it is used. For this reason open subroutines are often not used where they might provide the most efficient method. For example, one wants a subroutine which clears a block of storage. Using the instruction STORE ZERO (STZ) which clears the designated storage word, the following simple routine could be constructed:

```
AXT   N, 1
STZ   A+N, 1
TIX   *-1, 1, 1
```

In the above routine N is the symbol for the number of words in the block, and A is the symbol for the first location in the block. If one wants to clear three blocks of, say, 100, 150 and 75 words starting with locations R, S and T, he could write

```
AXT   100, 1          TIX   *-1, 1, 1
STZ   R+100, 1        AXT   75, 1
TIX   *-1, 1, 1       STZ   T+75, 1
AXT   150, 1          TIX   *-1, 1, 1
STZ   S+150, 1
```

C. SUBROUTINE WRITE-UP

It should be expected that every subroutine would be described properly, since a program cannot be considered as a subroutine unless there is an accompanying useful write-up. Certainly any device or tool, regardless of expense, is truly worthless if one cannot figure out the way to operate it. Such a tool can be damaging if improperly used. Nevertheless, every year finds too large a number of subroutines, many of which cost a considerable amount of money to prepare, being discarded because of inadequate write-ups and inadequate up-dating. Often the description accompanying the program is accurate only for an earlier version of the program.

A proper solution to such waste is the establishment of a set of standards. The good write-up must impart all the required information to the user in an easily digestible form. Furthermore, it must avoid the inclusion of irrelevant material which will obscure the meaningful. Is it possible to determine a set of rules and regulations that will effectively direct the programmer to create such an adequate write-up? Programming associations, e.g., SHARE, have always considered the question of write-up standards one of their prime reasons for existence. Such organizations can establish rules and refuse to distribute material which does not conform to the rules, although these steps are not necessarily sufficient to solve this problem. The priceless ingredient, which rules cannot supply, is the programmer's attitude.

D. FUNCTION STATEMENTS

Testing procedures greatly increase the programmer's efficiency in writing programs. The other main simplification that may be used in preparing programs is the incorporation of prewritten subprograms that may effect operations that are used repeatedly. For example, if one needs the value of the square root of a quantity, he may direct the computer to calculate this as part of his program. However, because this operation is so frequently carried out, it is much more efficient to write the steps of the operation in the form of a subprogram that could be called by the main program whenever needed. This is the purpose of subroutines and function definitions.

Subprograms (or subroutines) are of two types: (1) those that are built into the computer as part of its library of programs, and (2) those supplied by the programmer as part of his program deck. The former type will be considered first because of its ready availability. We need to only know the form by which the required subprogram is called. A typical example follows:

```
A = SQRTF (B)
```

If the above statement is incorporated into a program, A will take the value of the square root of B. This is an example of a function statement. All function names contain six or less alphabetic or numeric characters; they do not start with a number and end with a letter (Wiberg, p. 37).

The function of definitions may be modified as part of a program (Kuo, p. 72). For example, one may wish to evaluate both

common and natural logarithms, whereas the built-in programs give only the natural logarithm. Before one calls for a logarithm the first time, he may write the statement

```
LOG10F(X) = 2.303 * LOGF(X)
```

and whenever the statement

```
A = LOG10F(X)
```

appears, the common logarithm would be calculated. Similarly, if one wished to deal with angles in degrees rather than in radians (as the subprograms do), he might include statements such as

```
SINAF(X) = SINF(X/57.2958)
ASINAF(X) = 57.2958 * ASINF(X)
```

in the program.

It is sometimes useful to write a FUNCTION type of subprogram. A program of this type starts with a statement such as

```
FUNCTION MAX (A,B)
```

Here the name of the function is MAX and its arguments are A and B. The function would be called in the main program by a statement such as

```
X = MAXF (A,B)
```

Suppose the function subprogram were to determine which of the two variables, A and B, were the larger. X would then be given the value of the larger of the two. The subprogram would be written:

```
      FUNCTION MAX (A,B)
      IF (A-B) 1, 2, 2
    1 MAX = B
      GO TO 5
    2 MAX = A
    5 RETURN
      END
```

Note that the name of the function in the preceding example must be set equal to the quantity to be returned to the main program. Thus, here, MAX is set equal to A or B, whichever is the larger. The last statement, RETURN, effects a return to the main program.

E. SUBROUTINE ASSEMBLY

In using library subroutines one must have a knowledge of the way they function (fixed point, floating point, etc.) and arrange to keep within the limits of their specifications and ranges. One must remember that the actual code for the subroutine is a function of its operating locations in storage. To be useful, a library subroutine must be designed to fit into any program, provided only that sufficient storage space is available to contain the instructions, parameters and intermediate results of the subroutine. The subroutine must be stored in such a way that a copy of it can be assembled in a correct form for operation in any storage region the coder wishes to specify. The assembly operation can be carried out easily by the computer itself, if proper conventions of address notation and format are employed in the file copy of the subroutine.

The computer will carry out the assembly operation under the control of an assembly program. Most computing laboratories will keep one or more assembly programs in the service library (Stein, p. 227). Assembly programs will most likely vary from one installation to another.

All subroutines to be assembled must be written in a proper form dictated by the particular assembly program to be used. This

will permit the assembly program to distinguish between those parts of the subroutine which will remain fixed and those which will be modified. That is, it must be able to discriminate between quantities which are independent of storage and operation locations (e.g., constant operands and fixed addresses) and those which are in a direct function of these locations.

F. ENTRY AND EXIT

Subroutines require an entry and exit. The entry of the subroutine may consist of a few commands which create an exit command and then transfer to the working part of the subroutine. The exit is a planted transfer of control command. A dummy transfer of control command is created by the entry of the subroutine and placed at the end of the subroutine. The entry and exit are locations where any routine has its sequence of operation interrupted. The entry to a subroutine is the location to which control is transferred from the exit of the main routine. The exit of the subroutine is the point where control is transferred from the end of the subroutine back to an entry to the main routine.

G. RELATIVE ADDRESS

When a subroutine is written the addresses used generally assume the first command of the subroutine located in cell zero. That is, all the addresses are written relative to cell zero. When the subroutine is actually used and the region in memory around cell zero is not available (certainly not more than one subroutine could be placed in the area of cell zero), it becomes

necessary to move the entire subroutine to some other location in the memory.

H. ACTUAL OR ABSOLUTE ADDRESS

The relative addresses of the subroutine merely have added the amount by which the subroutine is to be shifted in memory, thereby becoming the actual or absolute addresses. Techniques for performing this shift are available within the actual hardware of the computer in some cases, or certain assembly routines are available which makes it possible to move these subroutines around.

I. RECORD COMMANDS

To facilitate the use of subroutines, record commands are available which enable the computer to store temporarily the address of the main program from which entry is made to the subroutine. It is then a part of the subroutine to take this stored address and create with it an exit from the subroutine back to the main program.

J. SUBROUTINE OBLIGATIONS

One of the first important obligations of a subroutine to a main program becomes apparent. The subroutine must do everything in its power to preserve the status of the computer as it is before the entry of the subroutine. If it is impossible or undesirable to perserve some part of the status, such as a register's contents or a switch position, notice must be given to the user.

Another obligation of the subroutine is to protect itself from

misuse in the sense of incorrect or insufficient data from the main program. The main program should be able to correct errors.

TYPES OF SUBROUTINES

A. OPERATING SUBROUTINES

Up to this point in this paper subroutines have been described as subservient to the main program, being given control at its pleasure. There are, however, sets of subroutines that partially control the computer through the trapping mechanisms, returning control to the main program only as their work is completed (Wegner, p. 188). Such subroutines are known as operating subroutines. The most prominent type of such routines is the input-output set, or "package."

Even though such packages usually consist of an elaborate and comprehensive set of subroutines for input, output, editing and conversion, mention here will be made relative to a much simpler package whose functions are restricted to the transmission of input and output data. The write-up of such an imaginary package would include: name, author, purpose, restrictions, calling sequence, method, physical description, timing, check-out procedures and a double check.

B. PROCESSING SUBROUTINES

A computer must have the ability to carry out the basic arithmetic operations of addition, substraction, multiplication and division. Nevertheless, without any detailed knowledge of these instructions, it is possible to carry out a great deal of

arithmetic through use of subroutines.

Of basic importance in mathematics is the concept of "function." A function is simply a rule for obtaining one set of numbers from another set. As such, the function bears an important relationship to the processing subroutine, which can be considered a "rule" by which one set of computer information is transformed into another set. Notice that this common use of the word "function" is perfectly consistent with the mathematical use. In fact, when mathematicians use tables, they speak of the "argument" (part number) and the "function" (price).

The importance of processing subroutines is that they enable us to carry out the rule implied by either a mathematical function, like square root or a nonmathematical function like withholding tax (McCalla, p. 34). Although one may not have the vaguest notion of what the "Gudermannian" of a number means, if given a subroutine and a calling sequence he can certainly calculate Gudermannians as well as anyone. The process of writing the necessary subroutines does, of course, require special knowledge in each case; but so does the process of building a computer.

According to Hassitt (p. 220), processing subroutines are not restricted to evaluating functions of single arguments nor are they restricted to yielding single results. Again, the means of using the subroutine are independent of the particular type of process.

USE OF SUBROUTINES IN A CHEMICAL LIBRARY

Perhaps the first question that comes to one's mind is that

relative to how does a subroutine being utilized for a chemical library purpose differ from any other subroutine? It is true that more similarities than differences occur among subroutines. However, in this case the nature of the library (chemistry-oriented) is unique in itself. In the following statements this writer briefly reveals how subroutines are used for such specific library purposes, plus explains some of the unique factors possessed by subroutines designed for chemical libraries.

Before going any further in thought along this line, it must be understood that the computer or computers will be operated in this particular library by a competent person possessing full understanding of the library's objectives and purposes.

Based on the foregoing information in this paper, it is evident that a subroutine will aid the librarian or chemist in solving a chemical equation. One may know what particular equation to use, but may have no idea as to the methods of solution for such equation. With the necessary subroutine, he may proceed without any difficulty.

Ideally, with the subroutines the computer's "knowledge" is expanded and it is able to do many more complex chemical related calculations, at the same time requiring less programming on the part of the user. The basic computations, which are of great value to the chemical field, performed by the subroutines include:

Basic Arithmetic

Floating point add, subtract, multiply, divide

Complex number operations

Multiple precision routines

Function Evaluation

Square root

Trigonometric: sin x, cos x, tan x

Exponential: ln x, log x, 10^x e^x, or a^x

Numerical Analysis

Solution of n simultaneous equations

Matrix operations

Eigenvalue solutions

Numerical integration

Numerical differentiation

Roots of polynomials

Logical

Sorting

Collating

Without further elaboration one can detect the significant contribution that the subroutines can make to computing chemical knowledge.

Effective roles are played by the service subroutines as compilers, assemblers, programming aids (e.g., debugging subroutines) and machine-testing routines. The functions performed by these service subroutines are numerous.

The up-to-date chemical library, depending on size and demand, may possess a "library within a library." Today it is very important to have a knowledgeable library of subroutines. Most of the subroutines will probably be maintained on a special-magnetic tape in lieu of punched cards which do not provide the same degree of convenience or speed as the tape.

CONCLUSION

What is information? Webster defines it as "Timely or specific knowledge acquired or derived; facts; data." From this long narration, but meaningful throughout, one is led to conclude that subroutines enlarge the importance of computers in regard to the handling of information. Naturally, much more than given in these past few pages exists on the significance of subroutines. However, the "highlights" have been presented. Advantages greatly outweigh the disadvantages of the function of subroutines.

What does the future hold for subroutines? Most of the world's knowledge will soon be in machine-readable form. It will be much like a game with subroutines; always interesting, occasionally frustrating, but never dull .... and the subroutine always wins (Rhynas, p. 20).

REFERENCES

Computers in 1984. Special Libraries, 56 (January, 1965), 58.

Cowan, Russell A. Computer Software: Problems and Solutions. Computer Design, 6 (September, 1967), 16-22.

Cutler, Donald I. Introduction to Computer Programming. Englewood Cliffs, New Jersey: Prentice-Hall, 1964.

Hassitt, Anthony. Computer Programming and Computer Systems. New York: Academic Press, 1967.

Hellerman, Herbert. Computer System Performance. New York: McGraw-Hill, 1975.

Kuo, Shan S. Computer Applications of Numerical Methods. Reading, Massachusetts: Addison-Wesley, 1972.

Lapidus, Leon. Digital Computations for Chemical Engineers. New York: McGraw-Hill, 1962.

London, Keith. Techniques for Direct Access. New York: Petrocelli Books, 1973.

McCalla, Thomas. Introduction to Numerical Methods and FORTRAN Programming. New York: Wiley, 1967.

Moon, B. Alno. Computer Programming for Science and Engineering. Chemistry and Industry, (July 22, 1967), 1248-49.

Musk, Fred I. One Man's Meat; It's All Good Grist That Comes to Our Mill. Computer Journal, 10 (August, 1967), 126-27.

Rhynas, Phillip. Computers and Programming. British Columbia Library Quarterly, 30 (April, 1967), 13-20.

Rosenbrock, Harold H. Computational Techniques for Chemical Engineers. New York: Pergamon, 1965.

Scott, Norman R. Electronic Computer Technology. New York: McGraw-Hill, 1970.

Stein, Marvin. Computer Programming. New York: Academic Press, 1964.

Trends and Forecast, '68 and Beyond; Computer Aid to Solve Intricate Problems. Product Engineering, 39 (January 29, 1968), 29-31.

Warheit, Ira A. The Principles of Computer Operation. Special Libraries, 51 (November, 1960), 485-492.

Wegner, Peter. Introduction to Symbolic Programming. New York: Hafner, 1963.

Wiberg, Kenneth B. Computer Programming for Chemists. New York: Benjamin, 1965.

Wrubel, Marshall H. Primer of Programming for Digital Computers. New York: McGraw-Hill, 1959.